New Agent Architecture for Evaluation in Goddard's Agent Concepts Testbed

Walt Truszkowski
NASA Goddard Space Flight Center
Code 588.0
Greenbelt, MD 20771
301-286-8821
Walt.Truszkowski@gsfc.nasa.gov

Christopher Rouff
NASA Goddard Space Flight Center
Code 588.0
Greenbelt, MD 20771
301-286-8821
Chris.Rouff@gsfc.nasa.gov

ABSTRACT

This paper presents the agent architecture that is being developed by the Goddard Space Flight Center as part of its program to investigate the role of agents and communities of agents in realizing autonomy of both ground-based and space-based systems. The agent architecture is component-based and each component and its dynamic behavior is described. An overview of an operational scenario that is being developed in the Agent Concepts Testbed (ACT) to evaluate the architecture is presented.

Keywords

Agent, architecture, component, behavior

1. INTRODUCTION

NASA has set for itself far-reaching autonomy goals for both its ground-based and space-based systems. More reliance on "intelligent" systems and less reliance on human intervention characterize its autonomy goals. These goals are further complicated by NASA's plans to use constellations of nanosatellites for future science data-gathering.

The Advanced Architectures and Automation Branch at the Goddard Space Flight Center has a leading role in the development of agent-based approaches required to realize NASA's autonomy goals. A past major success at Goddard was the development of a multiagent system called LOGOS (Lights-Out Ground Operations System) [2,3]. LOGOS provided an initial insight into the power of communities of agents supporting ground systems operations. Based on the success of this first prototype development has begun on the Agent Concepts Testbed (ACT), an environment in which richer agent and agent-community concepts will be evaluated through detailed prototypes and comprehensive operational ground-based and space-based scenarios. This paper addresses the new agent architecture that will be developed and evaluated in the ACT. It also presents a brief overview of the operational scenario, involving a community of agents, which will be implemented in the ACT.

2. OVERVIEW OF THE AGENT ARCHITECTURE

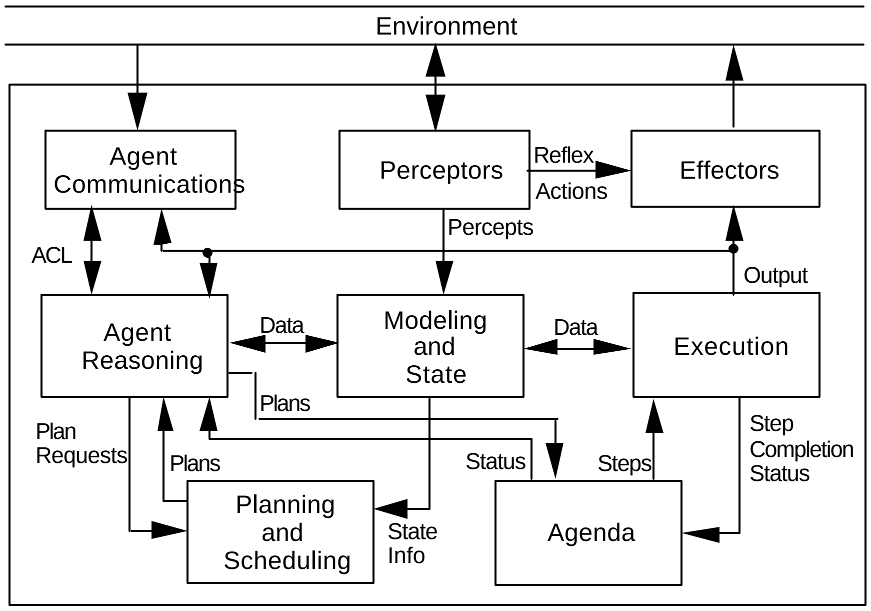
A new agent architecture has been introduced for evaluation in the Agent Concepts Testbed (ACT). The new agent architecture is a component-based architecture which allows greater flexibility to the agent designer.

A simple agent can be designed by using a minimum number of components that would receive percepts (inputs) from the environment and react according to those percepts. This type of simple agent would be a reactive agent.

A robust agent may be designed using more complex components that allow the agent to reason in a deliberative, reflexive and/or social fashion. This robust agent would maintain models of itself, other agents in its environment, objects in the environment that pertain to its domain of interest, and external resources that it might utilize in accomplishing a goal. Figure 1 depicts the components for a robust agent. The identified components give the agent a higher degree of intelligence when interacting with its environment.

Percepts received through sensors, communication with external software/systems, and other environmental entities are received through a Perceptor component. These percepts are passed from the Perceptor to the Modeling component where a model's state is updated as needed. (Note, the modeling component maintains models of objects of interest in the environment, other agents in the community, external resources and the agent itself.) A special Perceptor is used to send and receive messages with other agents, this is the Agent Communication Perceptor/Effector. Incoming Agent Communication Language (ACL) messages are formatted and passed to the Reasoning component.

The Agent Reasoning component reasons with received ACL messages, knowledge that it contains, and information that is



Features Include:
- Component-based architecture
- Distributed knowledge bases
- Publish/subscribe mechanism for information sharing amoung components
- Capable of reactive, deliberative, and social actions
- Can support multi-modal reasoning (rules, cases, models)

Benefits include:
- Robust infrastructure
- Easy to update
- Easy to tailor to a domain
- Supports reuse

Figure 1: ACT Agent Architecture

acquired from the Modeling component to formulate goals for the agent when necessary. Goals are then acquired by the Planning component along with state and state transition information.

The Planning component formulates a plan for the agent to achieve the desired goals. When a plan has been developed, the Agenda keeps track of the execution of the plan's steps. Steps are marked when they are ready for execution and the completion status of each step is also tracked by the Agenda.

The Execution component manages the execution of steps and determines the success or failure of each step's execution. Output produced during a step execution can be passed to an Effector or the Reasoning component. The Modeling component will record state changes caused by a step execution. When a plan is finished executing, the Agenda component sends a completion status to the Reasoning component to indicate that the goal established by the Reasoner has been accomplished. If the Agent Reasoning component is dealing with data from the environment it may decide to either set a goal (for more deliberative planning) or react quickly in an emergency situation. The Reasoner can also carry on a dialog with another agent in the community through the Agent Communication Perceptor/Effector.

The agent architecture is component-based. What is a component? A component is a software module that performs a defined task. Components when combined with other software components can constitute a more robust piece of software that is easily maintained and upgraded. Each component in the architecture can communicate information to/from all other components as needed through various mechanisms including a publish and subscribe communication mechanism, message passing, or a request for immediate data.

Components may be implemented with a degree of intelligence through the addition of reasoning and learning functions. Each component needs to implement certain interfaces and contain certain properties. Components must implement functionality to publish information, subscribe to information, and be able to accept queries for information from other components or external resources being used by the component. Components need to keep a status of their state, and need to know what types of information they contain and need from external components and objects to function.

Goddard's emphasis has been on the development of agent communities (rather than a large monolithic agent) to realize the desired levels of intelligent autonomous behaviors. Community-based approaches require an agent communication language to support dialogs among the agents in the community. The agent communication language under development at Goddard is based on the Agent Communication Language (ACL) of FIPA (Foundations of Intelligent Physical Agents - an international organization devoted to establishing standards for agent development) [1].

The agent architecture described above is capable of several types of behaviors. Basically, "agent behavior" refers to the manner in which an agent responds to some sort of stimulus, either externally (outside the agent) or internally (within the agent) generated. We have identified four basic classes of behaviors for our agent to realize. These are:

? social,

? proactive,

? reactive and

? deliberative.

These are further broken down by the source of the stimulus for the behavior. The current list of behavior types is:

? social - triggered by another agent

? social - triggered by the agent itself

? proactive - self motivating

? reactive - triggered by another agent

? reactive - triggered by a percept

? deliberative - triggered by another agent

? deliberative - triggered by a percept

What follows is a brief high-level definition of each of the identified classes of behavior.

Social.

Social behaviors refer to behaviors shared between/among agents. The current agent architecture supports two types of social behavior, i.e., social behavior triggered by another agent and social behavior triggered by the agent itself. In each of these cases the agent utilizes ACL messages to solicit help or to coordinate the behaviors of other agents.

Proactive

This type of behavior is that which is stimulated in some way by the agent itself. For our agents there is one type of proactive behavior that will be supported, i.e., self motivating. Self-motivating behaviors are triggered by built-in or intrinsic goals.

Reactive

Reactive behaviors are those that require "no thinking'. These behaviors are like built-in reflexive actions that are triggered by events in the agent's environment that are detected by the agent. When detected the agent responds immediately with a predetermined action.

Deliberative

This type of behavior is perhaps the most difficult and interesting. At the highest level of abstraction this type of behavior involves the establishing of an hierarchy of goals and subgoals, the development of plans to achieve the subgoals, and the execution of the planned steps to ultimately accomplish the goal which started the process of deliberation in the first place.

3. ARCHITECTURE COMPONENTS

As stated above, agents in ACT are built using a component architecture. A component is a software module that performs a defined task. Components when combined with other software components can constitute a more robust piece of software that is easily maintained and upgraded. Components can be easily swapped out and replaced by another more advanced component. These components were introduced briefly above. What follows is a more detailed look at each of them.

3.1 Modeler

The modeling component is responsible for maintaining the domain model of an agent, which includes models of the environment, other agents in the community, and the agent itself. The Modeler receives data from the Perceptors and agent communication component. This data is used to update state information in its model. If the data causes a change to a state variable the Modeler then publishes this information to other components in the agent that have subscribed to updates to that state variable. The Modeler is also responsible for reasoning with the models to act proactively and reactively with the environment and events that affect the model's state. In the future the Modeler will also dynamically modify its model based on experience.

The modeler can also handle what-if questions. These questions would primarily come from the planning and scheduling component, but may also come from other agents or from a person who wants to know what the agent would do in a given situation or how a change its environment would effect the values in its model.

3.2 Reasoner

The Reasoner component works with information in its local knowledge base as well as model and state information from the Modeler to make decisions and formulate goals for the agent. This component reasons with state and model data to determine if any actions need to be performed by the agent to effect its environment, change its state, perform housekeeping tasks, or other general activities. The Reasoner will also interpret and reason with agent-to-agent messages that are received by the agent's communications component. When action is necessary for the agent, the Reasoner will produce goals for the agent to achieve. Currently the Reasoner works more in a reactive manner. Either an input coming in or a trigger from the clock sets it in motion. Work is also being done to make the Reasoner more proactive.

The Reasoner currently uses a rule base to do it's reasoning, but is being expanded to include a case based and in the future a model based Reasoner, and neural net.

3.3 Planner/Scheduler

The Planner/Scheduler component is responsible for any agent level planning and scheduling. The planning component is given a goal or set of goals to fulfill in the form of a plan request. This typically comes from the Reasoning component but may be generated by any component in the system.

At the time that the plan request is given, the planning and scheduling component acquires a state of the agent and system, usually the current state, as well as the set of actions that can be performed by this agent. This information will typically be acquired from the modeling and state component. The planning and scheduling component then generates a plan as a directed graph of steps. A step is composed of preconditions to check, the action to perform, and the expected results from the action (post condition). When each step is created, it is passed to any Domain

Expert components/objects for verification of correctness. If a step is deemed incorrect or dangerous, the Domain Expert may provide an alternative step, solution, or data to be considered by the planner.

Once the plan is completed, it is passed back to the component that requested the plan (usually the Reasoner). The requesting component then either passes it on to the Agenda to be executed or uses it for planning/what-if purposes.

3.4 Agenda/Executive

The Agenda and Executive work together to execute the plans developed by the Planner/Scheduler. The agenda typically receives a plan from the Reasoner, though it can receive a plan from another component that is acting in a reactive mode. The agenda interacts with the Execution component to send the plan's steps in order, for execution. The agenda keeps track of which steps are being executed, finished executing, idle, or waiting for execution. It updates the status of each step appropriately as the step moves through the execution cycle. The agenda reports the plan's final completion status to the Planner and Agent Reasoner when the plan is complete.

The Executive executes the steps it receives from the Agenda. A step contains preconditions, an action and possible post-conditions. If the preconditions are met, the action is executed. When executions finish, the post-conditions are evaluated, and a completion status is generated for that step. The completion status is returned to the agenda, which allows for overall plan evaluation.

The execution component interacts with the agenda in the following way. The agenda sends the first step to the execution component. This wakes the Executive up. The component then begins executing that step. The Executive then checks to see if another step is ready for execution, if not, the component will go back to sleep until it receives another step from the agenda, once all executing steps are completed.

A watch is also attached to the executive that monitors given conditions during execution of a set of steps and a consequence if the condition occurs. Watches allow the planner to flag things that have to be particularly looked out for during real-time execution. They can be used to provide "interrupt" capabilities within the plan. An example of a watch may be to monitor drift from a guidance star while doing an observation. If the drift is over a threshold, then the observation is halted. In such a case the watch would notify the Executive which in turn would notify the Agenda. The Agenda would then inform the Reasoner that the plan failed and the goal was not achieved. The Reasoner would then formulate another goal (e.g., recalibrate the star tracker).

3.5 Agent Communications

The agent communication component is responsible for sending and receiving messages to/from other agents. The component takes an agent data object that needs to be transmitted to another agent and converts it to a message format understood by the receiving agent. The message format that is being used is based on FIPA. The message is then transmitted to the appropriate agent though the use of a NASA developed agent messaging protocol/software called Workplace [4].

The reverse process is performed for an incoming message. The communications component takes the message and converts it to an internal agent object and sends it out to the other components that are subscribing to incoming agent messages. The communications component can also have reactive behavior where for a limited number of circumstances it produces an immediate response to a message.

3.6 Perceptors/Effectors

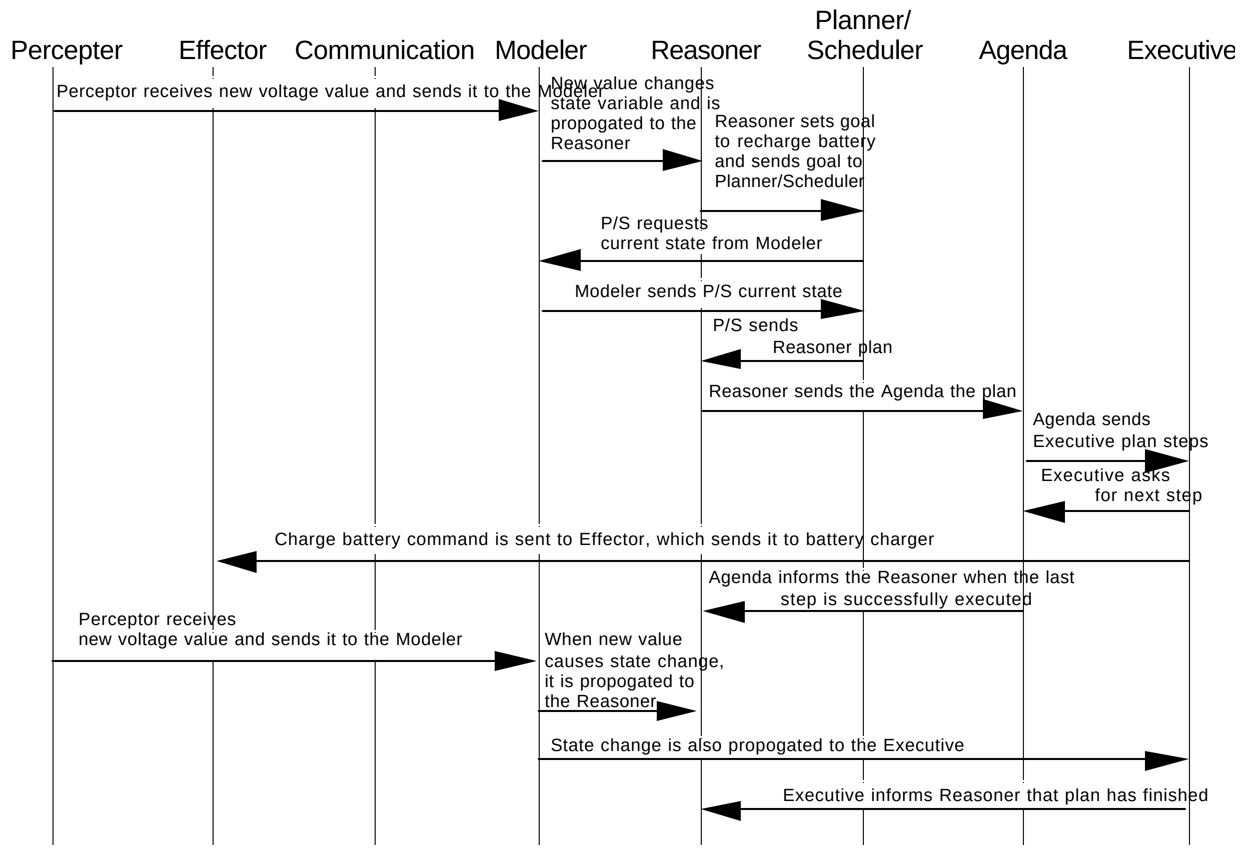
The Perceptors are responsible for monitoring parts of the environment for the agent. An example of what an agent might monitor is a subsystem of a spacecraft. Any data received by the agent from the environment, other than agent-to-agent messages, enters through Perceptors. An agent may have zero or more Perceptors, where each Perceptor receives information from specific parts of the agent's environment. A Perceptor may just receive data and pass it on to another component in the agent or it may perform some simple filtering/conversion before passing it on in the agent. A Perceptor may also act intelligently through the use of reasoning systems if it is desired. If an agent is not monitoring a part of the environment, then it would not have any perceptors (an example of this would be an agent that only provides expertise in a certain area, such as fault resolution).

The Effector is responsible for effecting or sending output to the agent's environment. Any agent output data, other than agent-to-agent messages, leaves through Effectors. Typically the data coming from the Effectors will be sent from the executive which has just executed a command to the agent's environment. There may be zero or more Effectors, where each Effector sends data to specific parts of the agent's environment. An Effector may perform data conversions when necessary and may even act intelligently and in a proactive manner when necessary through the use of internal reasoning systems if it is desired. As with the Perceptors, an agent may not have an Effector if it is not interacting with the environment.

3.7 Agent Framework

A framework is used that the components are plugged into that provides a base functionality for the components as well as the inter-component communication functionality. The framework allows components to be easily added and removed from the agent while providing for a standard communications interface and functionality across all components. This makes developing and adding new components easier and makes the addition transparent to existing components in the agent.

The communications mechanism for components is based on a publish and subscribe model with direct links between components when large amounts of data need to be transferred. Components communicate to each other the types of data that it



Percepter
Effector
Communication
Modeler
Reasoner
Planner/ Scheduler
Agenda
Executive
Perceptor receives new voltage value and sends it to the Modeler
New value changes state variable and is propogated to the Reasoner
Reasoner sets goal to recharge battery and sends goal to Planner/Scheduler
P/S requests current state from Modeler
Modeler sends P/S current state
P/S sends Reasoner plan
Reasoner sends the Agenda the plan
Agenda sends Executive plan steps
Executive asks for next step
Charge battery command is sent to Effector, which sends it to battery charger
Agenda informs the Reasoner when the last step is successfully executed
Perceptor receives new voltage value and sends it to the Modeler
When new value causes state change, it is propogated to the Reasoner
State change is also propogated to the Executive
Executive informs Reasoner that plan has finished

Figure 2: Scenario of data flowing between agent components.

produces when queried. When one component needs to be informed of new data or changed data in another component it subscribes to the particular data in the component in which it is interested. Data can be subscribed to whenever it is changed or on an as needed basis. With this mechanism a component can be added or removed without having to modify the other components in the agent.

4. DATAFLOW BETWEEN COMPONENTS

This section gives an example of how data flows between components of the architecture. The example being used is when a spacecraft's battery is discharging. Figure 2 shows a timeline and the flow of data between components. The scenario reads as follows:

? The agent detects the low voltage by reading data from the battery via a Perceptor. The Perceptor then passes the voltage value to the Modeler, which has subscribed to the Perceptor to receive all percepts.

? When the Modeler receives the voltage from the Perceptor, it converts the voltage data to a discrete value and updates this value in the model. In this case, the updated voltage value puts it below the acceptable threshold and changes the model's voltage state to "low". This change in state value causes a state change event and the Modeler now publishes the new state value to all components that have subscribed to changes in this state variable. Since the Reasoner has subscribed to changes in this state variable, the low data value is sent to the Reasoner.

? In the Reasoner the low voltage value fires a rule in the expert system. This rule calls a method that sends the Planner/Scheduler a goal to achieve a battery voltage level that corresponds to fully charged.

? When the Planner/Scheduler receives the goal from the Reasoner, it queries the Modeler for the current state of the satellite and a set of actions that can be performed (this set may change based on the health of the satellite).

? After receiving the current state of the satellite and the set of available actions from the Modeler, the Planner/Scheduler formulates a list of actions that need to take place to charge the battery. It then sends the plan back to the Reasoner for validation.

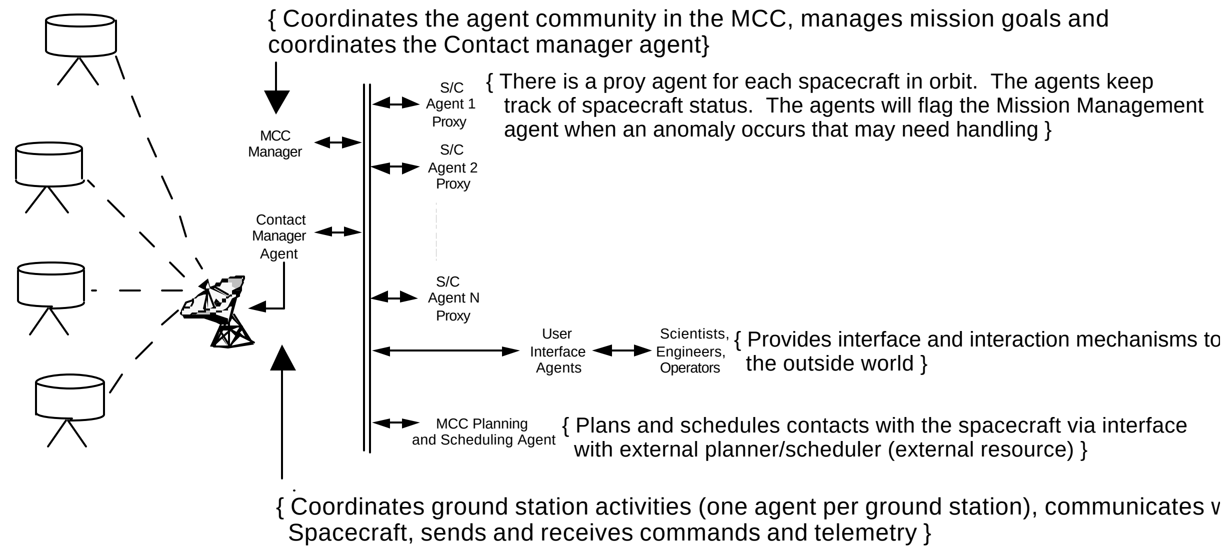

{ Coordinates the agent community in the MCC, manages mission goals and coordinates the Contact manager agent}
MCC Manager
Contact Manager Agent
S/C Agent 1 Proxy
S/C Agent 2 Proxy
S/C Agent N Proxy
{ There is a proy agent for each spacecraft in orbit. The agents keep track of spacecraft status. The agents will flag the Mission Management agent when an anomaly occurs that may need handling }
User Interface Agents
Scientists, Engineers, Operators
{ Provides interface and interaction mechanisms to the outside world }
MCC Planning and Scheduling Agent
{ Plans and schedules contacts with the spacecraft via interface with external planner/scheduler (external resource) }
{ Coordinates ground station activities (one agent per ground station), communicates w Spacecraft, sends and receives commands and telemetry }

Figure 3: Agent community being developed in ACT to test out the new agent architecture and some community concepts.

? The Reasoner examines the set of actions received from the Planner/Scheduler and decides that it is reasonable. The plans are then sent to the Agenda.

? The Agenda then puts the action steps from the plan into a queue for the Executive.

? As the Executive is ready to execute a new step, the agenda passes them one at a time to the Executive for execution.

? The Executive executes each action until the plan is finished. At this time the Executive notifies the Agenda that it has finished executing the plan.

? The Agenda marks the plan as finished and notifies the Reasoner (or who ever sent the plan) that the plan finished successfully.

? After the plan is executed, the voltage starts to rise and will trigger a state change in the Modeler when the voltage goes back into the fully charged state. At this time the Reasoner is again notified that a change in a state variable has occurred.

? The Reasoner then notes that the voltage has been restored to the fully charged state and marks the goal as accomplished.

5. ACT OPERATIONAL SCENARIO

The agents architecture described above will be evaluated along with some agent community concepts in ACT this coming year. The operational scenario that has been developed is loosely based on some nanosatellite constellation ideas.

Figure 3 graphically illustrates this scenario. It is based around the idea of a ground-based community of proxy agents (each representing a spacecraft in the nanosatellite constellation) which provide for autonomous operations of the constellation. Future scenarios will depict the migration of this community of proxy agents to the spacecraft themselves for an evaluation of space-based autonomy concepts.

In this scenario there are several nanosatellites in orbit collecting magnetosphere data. The Mission Control Center (MCC) makes contact with selected spacecraft according to its planned schedule when the spacecrafts (S/C's) come into view.

The agents that would make up the MCC would be:

?? Mission Manager Agent: coordinates the agent community in the MCC, manages mission goals and coordinates Contact Manager Agents.

?? Contact Manager Agent: coordinates ground station activities (one agent per ground station), communicates with the spacecraft, sends and receives data, commands, and telemetry.

?? User Interface: interfaces with the user to accept commands for the spacecraft and sends data to be displayed.

?? MCC Planning/Scheduling Agent: plans and schedules contacts with the spacecraft via interface with external planner/scheduler.

?? Spacecraft Proxy Agents: there is a proxy agent for each spacecraft in orbit. The agents keep track of spacecraft status, health and safety, etc. The agents will notify the Mission Manager Agent when an anomaly occurs that may need handling.

An example of a typical contact with a satellite would be:

?? The Contact Manager Agent (CMA) receives an acquisition of signal (AOS) from a spacecraft. The MCC is now in contact with the spacecraft.

?? The CMA requests the S/C to start downloading its telemetry data. When the telemetry is downloaded, it is sent to a spacecraft proxy agent.

?? The proxy agent updates the state of its model of the spacecraft from the telemetry received. If a problem exists, a flag is raised to the Mission Manager Agent and appropriate action (if any) is planned by the system.

? The Contact Manager Agent analyzes the downloaded telemetry data. If the telemetry indicates a problem the CMA may alter the current contact schedule to deal with the problem.

? The CMA executes the contact schedule to download data, delete data, or save data for a future pass. For example, the commands: download packet1, download packet3, delete packet2, save packet4.

? The CMA performs any necessary commanding in parallel to doing any data downloads.

? The Mission Manager Agent ends contact.

6. CONCLUSION

The ACT agent architecture provides for a flexible implementation of a wide range of intelligent or reactive agents for NASA spacecraft and ground systems. It allows for easy removal of unneeded components for reactive agents and the inclusion of the necessary components to implement intelligent agents. It is also flexible so that additional unforeseen components that will implement new AI technologies can be added as they become available without effecting previously implemented components.

The ultimate goal of our work is to be able to transition proven agent technology into operational NASA systems. The implementation of the scenario discussed above (and others under development) in the ACT will provide an opportunity to exercise and evaluate the capabilities supported by the agent architecture and refine the architecture as required. It will also provide an opportunity for space mission designers and developers to "see" agent technology in action. This will enable them to make a better determination of the role that agent technology can play in their missions.

7. ACKNOWLEDGMENTS

We would like to acknowledge the other members of the ACT development team at Goddard, which include James Rash, Tom Grubb, Troy Ames, Carl Hostetter, Jeff Hosler, Matt Brandt, Dave Kocur, Kevin Stewart, Jay Karlin, Victoria Yoon, Chariya Peterson, and Dave Zock (who are or have been members of the Goddard Agent Group) for their major contributions.

8. REFERENCES

[1] Foundation for Intelligent Physical Agents (FIPA). FIPA Specification Part 2: Agent Communication Language. Geneva, Switzerland. November 28, 1997.

[2] LOGOS Overview, Design and Specification Documents. http://agents.gsfc.nasa.gov/products.html.

[3] Truszkowski, W., and Hallock, L. Agent Technology from a NASA Perspective. CIA-99, Third International Workshop on Cooperative Information Agents, Uppsala, Sweden, 31 July – 2 August 1999, Springer-Verlag.

[4] LOGOS System Overview Document. http://agents.gsfc.nasa.gov/documents/code588/LOGOS.stuff/logosoverview.pdf